J Sign Process Syst (2013) 72:165–180
DOI 10.1007/s11265-013-0784-9

Eco-Storage: A Hybrid Storage System with Energy-Efficient Informed Prefetching

Maen M. Al Assaf · Xunfei Jiang · Mohamed Riduan Abid · Xiao Qin

Received: 1 November 2012 / Revised: 14 March 2013 / Accepted: 20 May 2013 / Published online: 30 June 2013

Abstract In this paper, we present an energy-aware informed prefetching technique called Eco-Storage that makes use of the application-disclosed access patterns to group the informed prefetching process in a hybrid storage system (e.g., hard disk drive and solid state disks). Since the SSDs are more energy efficient than HDDs, aggressive prefetching for the data in the HDD level enables it to have as much standby time as possible in order to save power. In the Eco-Storage system, the application can still read its on-demand I/O reading requests from the hybrid storage system while the data blocks are prefetched in groups from HDD to SSD. We show that these two steps can be handled in parallel to decreases the system's power consumption. Our Eco-Storage technique differs from existing energy-aware prefetching schemes in two ways. First, Eco-Storage is implemented in a hybrid storage system where the SDD level is more energy efficient. Second, it can group the informed prefetching process and quickly prefetch the data from the HDD to the SSD to increase the frequent HDD standby times. This will makes the application finds most of its on-demand I/O reading requests in the SSD level. Finally, we develop a simulator to evaluate our Eco-Storage system performance. Our results show that our Eco-Storage reduces the power consumption by at least 75 % when compared with the worst case of non-Eco-Storage case using a real-world I/O trace.

Keywords Informed prefetching · Power consumption · Parallel storage systems · Hybrid storage system

M. M. Al Assaf (✉)
King Abdullah II School for Information Technology,
The University of Jordan, Amman, Jordan
e-mail: m_alassaf@ju.edu.jo

X. Jiang · X. Qin
Department of Computer Science and Software Engineering,
Auburn University, Auburn, AL 36849-5347 USA

X. Jiang
e-mail: xunfei.jiang@gmail.com

X. Qin
e-mail: xqin@auburn.edu

M. R. Abid
Alakhawayn University in Ifrane, Ifrane, Morocco
e-mail: r.abid@aui.ma

1 Introduction

Prfetching techniques are well known for solving the I/O bottleneck problem (see [30] and [19]) in data-intensive computing systems. A wide rang of prefetching techniques have been proposed by researchers to preload data from disks into the main memory prior to the data accesses. Prefetching techniques are mainly categorized into two types—predictive and informed prefetching. Informed prefetching makes use of the applications ability to disclose hints about their future I/O accesses to prefetch the data before it is actually accessed by the application [3]. Predictive prefetching schemes predict future I/O access patterns based on historical I/O accesses of applications [4]. Recent studies show that prefetching paradigm is not only used to solve the I/O bootlenick but also to create an energy-aware storage system [34]. These studies implies that hard disk drives (HDDs) consume too much power during both active and idle modes when they are compared to the standby (sleeping) mode. In case the HDDs is set to the standby mode or powered off frequently, the system's power consumption will be reduced. Aggressive prefetching process from the HDD to the main memory will makes the

HDD to standby frequently for long time intervals and save power. However, memory size and prefetching accuracy are the most important factors that helps in saving the power consumption. Another study [40] shows that solid state drives (SSDs) are more energy efficient than HDDs. For this reason, prefetching can be used effectively in hybrid storage system to prefetch the data from the HDDs level to the SDDs one in order to reduce the HDD power consumption.

When it comes to informed prefetching, the applications are ale to disclose accurate hints about their future accesses. These hints can be used to prefetch the data in groups from the HDDs level to the SSDs in a hybrid storage system.

In this study, we focus on an informed prefetching scheme that prefetch the data in groups and investigate its energy saving impact on a hybrid storage system.

1.1 Motivations

As the HDDs energy consumption is significant, relying on an energy efficient storage devices (e.g. SDDs) becomes important. Recent studies show that prefetching can reduce the HDDs power consumption [34]. In a hybrid storage system that consists of HDDs and SSDs, informed prefetching can aggressively prefetch the application's future hinted accesses from the HDDs to the SSDs to extend the HDDs frequent standby times intervals. This reduces the system's power consumption.

The following key factors motivate us to investigate our energy-aware informed prefetching:

1) the growing needs of hybrid storage systems,
2) The SSDs energy efficiency when compared to the HDDs,
3) the I/O access hints offered by applications, and
4) the possibility of prefetching mechanisms and application on-demand I/O reading requests to work in parallel.

1.2 Contributions

The following list summarizes the major research contributions made in this paper:

- To reduce the power consumption in a hybrid storage systems, we propose new energy-aware informed prefetching approaches to aggressively prefetch the data from an energy inefficient level (e.g. HDD) to an energy efficient level (e.g. SSD). This enables the energy inefficient level to sleep as much as possible. The informed prefetching algorithm developed in this study is called Eco-Storage. We show that with our prefetching mechanism in place, aggressive prefetching operations in form of groups can be processed in a hybrid storage system in parallel with the application on-demand I/O reading requests.
- We apply a novel of energy-aware cost-benefit model to estimate the value of prefetching a future hinted data block at a specific storage level in a hybrid storage system. The cost-benefit model is used by prefetching mechanisms deployed in a hybrid storage levels to reduce the power consumption.
- We develop a simulated hybrid storage system, in which the Eco-Storage prefetching technique is implemented. Simulation results show that our energy-aware informed prefetching mechanism reduces the system's power consumption.

1.3 Roadmap

The rest of the paper explains and justifies our energy-aware informed prefetching scheme in a hybrid storage system. Section 2 reviews the related work. We outline the Eco-Storage architecture in Section 3. Section 4 describes the design and implementation issues of the Eco-Storage energy-aware prefetching mechanism. Section 5 presents our experimental framework and results. Finally, Section 6 provides conclusions and directions for future studies.

2 Related Work

Previous researchers have suggested that prefetching paradigm can be used to dramatically decrease the energy consumption. To our best knowledge, however, ours is the first study to focus on energy-aware informed prefetching in a hybrid storage and the first to offer a systematic performance evaluation.

2.1 Hybrid Storage Systems

A hybrid storage system consists of a hierarchy of heterogeneous storage devices that differ in their hardware, speed, size, energy consumption, and other specifications [32]. Hybrid storage systems provide cost-effective solutions for large-scale data centers without significantly affecting I/O response times. Usually, the high-level are more energy efficient. For example if a hybrid storage system consists of a HDD at the bottom and SSD at the top, the top level is more energy efficient [40].

Typical storage devices in a modern hybrid storage system include main memory, solid state disks, hard disks, and magnetic tape subsystems (see, for example, [13, 14]).

Similar to multi-level caches, a Hybrid multi-level storage systems [11, 12] first check their upper-level storage devices. If data items are not found in the upper level

storage, the next storage level is checked then. This process is repeated until the required data block is found.

2.1.1 Solid State Disks and Hard Drives

Compared to traditional hard drives (HDD), solid state disks (SSD) show better data read performance and less energy consumption [40]. Tuma provides comparisons between a wide range of SSDs and HDDs; detailed comparisons can be found in [31].

2.1.2 Parallel Storage Systems

Parallel storage systems are commonly deployed in supercomputers [3, 13, 28]. A parallel storage system consist of redundant storage devices that offer high I/O performance and bandwidth [24, 27]. Disk arrays are considered the most important components in the parallel storage systems. Scalable parallel storage systems (see, for example, [13, 23, 28]) provide I/O parallelisms through the stripping technique, in which each large data block is cut into small data blocks and stripped among an entire disk array [29]. When a particular data block is requested, the request will be directed to the entire disk array, where multiple disk are controlled by a disk controller that can coordinate and respond to the request in parallel.

Parallel storage systems are reliable due to the fault tolerance features offered by data redundancy [25, 26].

Parallel hybrid storage systems can be save energy when energy efficient storage devices are used such as SSDs [40].

2.2 Informed Prefetching

A study conducted by Patterson *et al.* [3, 7, 8, 10, 16] inspired us to concentrate on informed prefetching issues. Patterson *et. al* described the informed prefetching algorithms that invokes the storage parallelisms by taking the advantage of the application-disclosed I/O access hints. Informed prefetching performs aggressive prefetching to helps in wrapping the application's I/O stalls [3, 8, 16]. The performance benefits of informed prefetching were also presented by other researchers such as Huizinga *et al.* [6] and Chen *et al.* [17].

When disk arrays are used, parallel informed prefetching aims to leverage parallel I/O to improve prefetching performance. Parallel informed prefetching is made possible, because data blocks are striped across an array of disks [15, 24]. Parallel informed prefetching eliminates I/O stalls by prefetching the data to the cache before it is actually requested by applications. Please note that when we use the word "cache" as a noun, we mean the portion in the application main memory address space that is reserved for caching the informed prefetching process. When we use the word "cache" as a verb, we mean the process of caching the informed prefetching process.

A informed prefetching mechanism makes prefetching decisions based on access hints and access patterns being given a priority. Researchers have investigated various ways of collecting information to offer accurate access hints for informed prefetching mechanisms. Without appropriate hints, the prefetching mechanisms are unable to make accurate decisions. Chang and Gibson proposed an application speculative execution scheme that pre-executes applications to record the applications future accesses [9]. Future access patterns recorded by Chang and Gibson's scheme can be used to guide informed prefetching algorithms to preload data that are likely to be accessed. Byna *et al.* studied a solution that combines post-execution and runtime analysis to reduce future I/O reads prediction overhead [20]. Byna's study is one step toward improving the performance of existing informed prefetching mechanisms. Our informed prefetching algorithm presented in this paper is orthogonal to the above research in the sense that integrating our solutions with these techniques of collecting hints can significantly reduce the power consumption for a hybrid storage systems.

2.3 Predictive Prefetching

Predictive Prefetching (a.k.a., automatic prefetching) relies on past I/O accesses history in order to predict the application's future accesses and to prefetch the predicted future data [4, 18]. One example of the predictive approach model is Griffioen and Appleton's. It was developed for encoding past access patterns and future access probabilities [4]. In their model, they used directed weighted graphs to estimate access probabilities [4].

Recently, we developed another predictive approach solution that models an automatic prefetching and caching system called APACS [18]. We integrated three unique techniques into APACS (1) dynamic cache partitioning, (2) prefetch pipelining, and (3) prefetch buffer management. Our solution—APACS dynamically partitions the buffer cache memory, used for prefetched and cached blocks, by automatically changing buffer/cache sizes in accordance to global I/O performance.

2.4 Energy-aware Prefetching and Caching

Several solutions used prefetching techniques to reduce the power consumption [34, 35]. Athanasios presented a novel method for reducing the storage system power consumption using prefetching and caching techniques. Their main concept is to make accurate prefetching decisions

in order to aggressively prefetch and cache the data from the disk. This makes the disk sleep as much as possible and save power. They tested their system performance using a wide variety of real world benchmarks. However, the system efficiency is restricted by the cache size and the prefetching process accuracy. Our solution differs from the existing energy-aware prefetching and caching techniques. It works for hybrid storage systems in the sense that it prefetch the data to an energy efficient storage device of the system based on the application-disclosed access patterns (prefetching process is always accurate).

3 System Design

Compared with existing energy-aware prefetching schemes, Eco-Storage introduces a set of salient features: support application-disclosed I/O access hints and can prefetch the data in a hybrid storage system in parallel with the application's on-demand I/O read requests. Before presenting the Eco-Storage implementation details, we first outline a high-level overview of its hardware and software architectures.

3.1 Hardware and Software Architectures

The system consists of an application (user) that can provide hints of its future accesses. The hierarchy from top to bottom consists of a cache and an array of two levels of storage devices (Solid State Drive (SSD) and Hard Disk Drive (HDD)) as shown in Fig. 1. SSD level is more energy-efficient than the HDD one.

Concerning Eco-Storage software architecture, the applications provide hints on future I/O accesses. Eco-Storage contains a software module that do informed prefetching process by fetching hinted blocks from hard drives to the solid state disks. Eco-Storage works in parallel with the application's on-demand I/O reading request that are issued to the hybrid storage system. Eco-Storage keeps prefetching a particular number—which depends on available I/O bandwidth and the desired number of prefetching groups—of hinted blocks to the SSD level. The application can use the cache to buffer its on-demand reads. In a hybrid storage system, the application's on-demand reading requests are first issued to solid state disks. If hinted blocks are not residing in the solid state disks, the blocks will be read from hard drives.

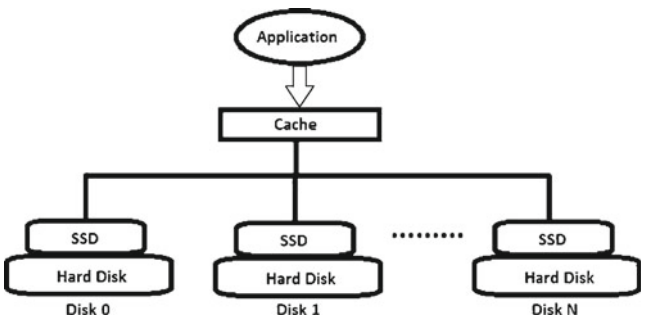

Figure 1 Eco-Storage's hardware architecture consists of an array of hybrid disks.

3.2 Assumptions

In this paper, we will assume a similar conservative assumption as we did in [1] and in [2]. We assume that hinted data blocks are initially allocated to HDDs thanks to the large capacity of the HDDs. It is noteworthy that I/O performance and energy efficiency of the hybrid storage systems can be improved if hinted blocks are initially placed in SSDs rather than HDDs.

In a hybrid storage system, a small portion of SSD space is reserved for retaining copies of the prefetched data. In order to save the I/O bandwidth, Eco-Storage keeps original copies at the HDD level while fetching duplicated copies to SSDs, instead of migrating data from HDDs to SSDs,

3.2.1 Bandwidth Limitations

We assume that the maximum I/O bandwidth offered by a parallel storage system equals the maximum number of concurrent read requests that may take place in the storage system without causing any congestion. Thus, the maximum I/O bandwidth depends on the scalability of the storage system. This means that each disk (storage node) can serve one reading request at a time without any congestion. A similar assumption can be find in the literature (see, for example, [3]). Striping patterns vary among the different parallel storage systems types (e.g. RAID various types). Our solution is valid whenever the storage system can provide parallelism. The study and evaluation of different striping patterns is out of our research scope. However, whatever stripping pattern is used, we stay on our assumption that the parallel storage system can serve a limited number of concurrent reading request at a time depending on the number of available disks.

3.3 Data Initial Placement

Let us consider a simple hybrid storage system, where there are two-level disk arrays composed of an SSD (top) and an HDD (bottom). The data initial allocation could be distributed between the two levels. In our simulation studies, we initially place all of the data blocks in the HDDs (bottom

level). This initial data placement is reasonable because of the following five reasons.

- HDDs have larger storage capacity than SSDs. So, it is more likely to find the data in the HDDs more than the SSDs.
- Research of prefetching in hybrid storage systems (see previous studies conducted by Nijim and Zhang *et al.*) indicates that the lower levels of the hierarchy contain the less important data [5, 13]. This implies that prefetching techniques comes to move data that is likely to be accessed in the future to the uppermost-level storage devices. This increases the probability of finding hinted data blocks in the HDDs level.
- To assume that all the data are initially allocated in the lowest level (e.g., HDDs in our study) represents the worst case scenario. This is because SSDs are more energy efficient than HDDs.
- In case that a hinted data block is already allocated in the SSD level, the corresponding informed prefetching request will be discarded.

3.4 Prefetching for Data Transfers

We build a prefetching model to transfer data to the top level in a hybrid storage system in parallel with the application on-demand read requests. In our design, the lowest-level storage devices (i.e. HDD) always achive original copies of prefetched data. Where the top-level (i.e. SSD) achieve copies of the original data. Our motivations behind this design are:

- Saving the higher-level storage devices space which are more expensive than the lower-level counterparts. Eco-Storage prefetching does not consume the higher-level storage space as we will see later.
- Migrating the entire data blocks from HDDs to SSDs needs to keep moving the data back and forth among multiple-level storage, which consumes I/O bandwidth.

3.5 Block Size

As we did in [1], hybrid storage systems consist of a storage devices hierarchy where the top level's performance is the best. Using SSDs and HDDs installed in the servers in our laboratory, we observed that an SSD does not provide better performance than an HDD for small data blocks; SSDs are faster than HDDs when the block size is at least 10MB. For this reason, we validated our system parameters based on that block size.

In addition, several file systems use large data block sizes to improve the I/O performance. They pack data into large blocks in order to achieve that. For example, HDFS (Hadoop Distributed File System) data block size is 64 MB [21]. In reference [22], HDFS block size is increased in order to obtain an improved system performance.In case the data blocks are small, Hadoop achieves (HAR) tool is used to pack them into a large one [33].

In case a different block size is used, our solution will stay valid and able to save energy. This is because it prefetches the data blocks from HDD (energy inefficient) to SDD (energy efficient). In addition to energy saving, our solution cares about achieving performance improvement. For this reason, in this paper we will use a 10MB block size.

3.6 LASR Traces

We developed a trace-driven simulators to evaluate performance of our Eco-Storage prefetching mechanisms under I/O-intensive workload. Traces used in our experiments represent applications that have negligible CPU processing time and a particular on-demand read time between each two subsequent I/O reading requests that we will validate later. All data blocks in the tested traces have identical block size. More specifically, we use the machine06 trace (called LASR2) [37] in our simulation study.

4 Eco-Storage Algorithm

The informed prefetching algorithm—TIP (see [3] for details)—makes use of applications' hints of future I/O accesses to prefetch the data before it is actually accessed. In this section, we will use the informed prefetching technique to bring the data in groups from the lower levels to the upper-most one in a hybrid storage system. This mechanism will be held in parallel with the application's on-demand I/O reading requests. Our hybrid storage system consists of two levels, SSD on the top and HDD in the bottom. In such system, SSD level is more energy efficient. The purpose of grouping the data prefetching process from the HDD to the SDD is to give the HDD as much as possible of standby time which reduces its energy consumption. At the HDD standby time, all the application's on-demand I/O reading requests will be found in the SDD level. We call our new informed prefetching technique Eco-Storage.

Our empirical experiments indicate that parallel storage systems may have I/O congestion. Evidence shows that there is a maximum number of read requests being concurrently processed in a parallel storage system. In the event that the application issues only one on-demand I/O reading request at a time, the rest unused I/O bandwidth can be allocated for our prefetching mechanisms to bring hinted blocks from lower-level to upper-level storage.

This section presents an algorithm that guides us in implementing the Eco-Storage mechanism for a single node parallel hybrid storage systems.

4.1 Definitions

Eco-Storage does informed prefetching between SSD level and the HDD one. When an application starts its execution, it starts to issue on-demand I/O reading request. Recall that we assume no caching. This means that the application continues to issue a single I/O reading request at a time. This may not utilize the parallel storage system's bandwidth. Based on the non-utilized portion of the I/O bandwidth, Eco-Storage assigns its prefetching depth.

Let Max_{BW} be the maximum number of read requests that may take place concurrently in the parallel hybrid storage system. Since each disk can handle at least one read request without congestion, the value of Max_{BW} depends on the number of disks nodes and the available aggregated I/O bandwidth. So, we will set Max_{BW} equals to the number of disks nodes. Since the application issues only a single on-demand I/O reading request to the parallel hybrid storage system, the rest bandwidth is used for Eco-Storage's prefetching depth. The prefetching depth is determined by Eq. 1; this number represents the maximum space that needs to be reserved for each group of Eco-Storage prefetching requests in the uppermost level (i.e, SSDs). Eco-Storage prefetching depth does not consume the SSD storage space

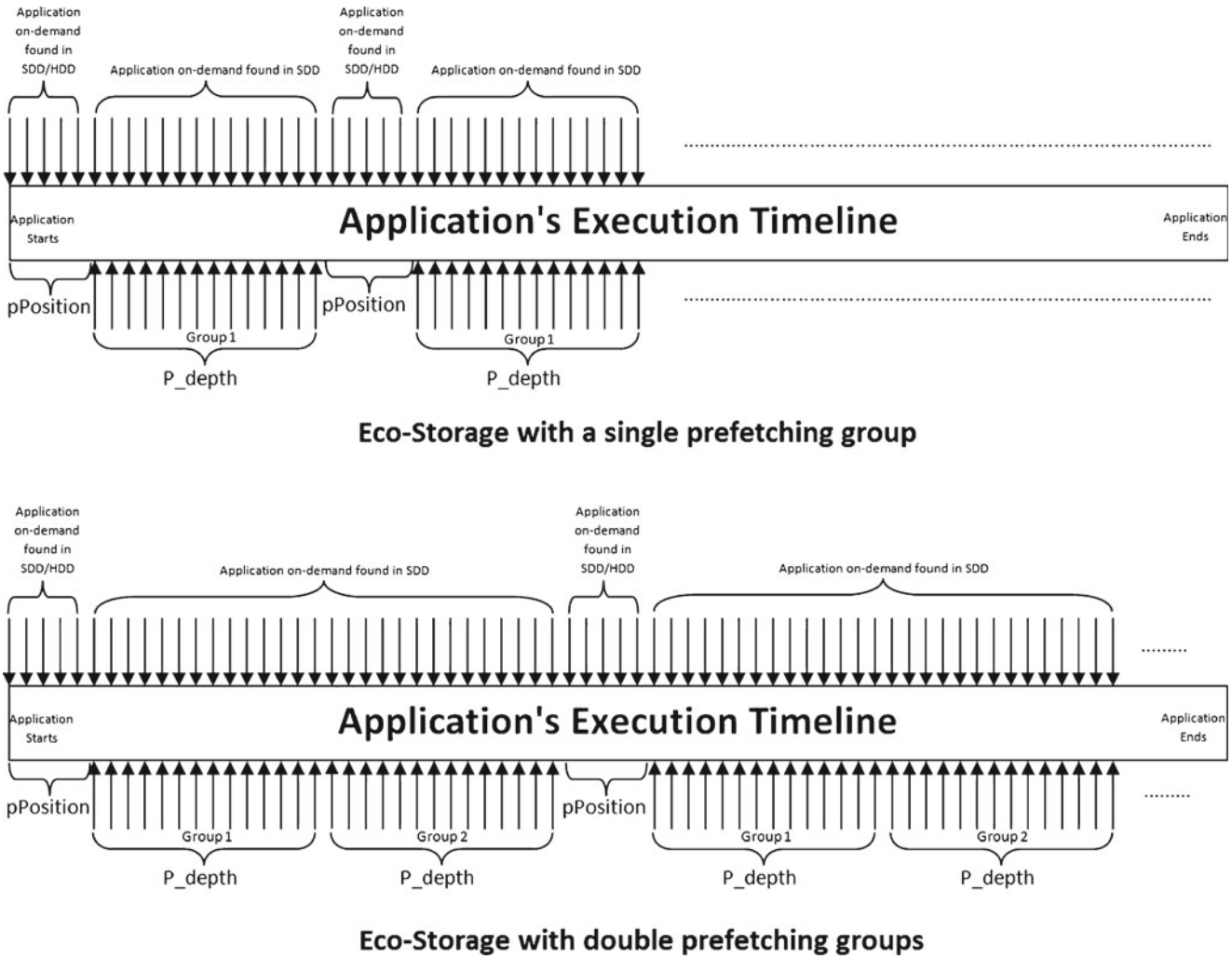

Figure 2 Eco-storage with single/ double prefetching group(s). When the application starts, it will continue to issue on-demand I/O reading requests. Eco-Storage starts to prefetch future hinted data blocks in groups from the HDD to the SSD. Each group is of P_{depth} depth. pPosition represents the distance between the consequent Eco-Storage grouped prefetching requests. When all the prefetched data blocks are accessed by the application, Eco-Storage issues new grouped prefething requests. When the number of prefetching groups increases, more on-demand requests will be satisfied from the SSD. This enables the HDD to sleep and to save power.

because it will reserve only a single data block space at each SSD of the array for each prefetching group. Please note that a prefetching group is a number of adjacent informed prefetching reads from the HDD to the SDD that equals to P_{depth})(see: Fig. 2).

$$P_{depth} = Max_{BW} - 1 \quad (1)$$

where;

P_{depth} Eco-Storage prefetching depth
Max_{BW} Maximum bandwidth

The application continues to issue a single on-demand I/O reading request to the hybrid storage system at a time. The request may be found in either levels. At the same time, Eco-Storage keeps fetching one or more future hinted data blocks groups from HDDs to the SSDs. Each group is of size P_{depth} hinted data blocks. Please note that the number of prefetching groups is arbitrary. Eco-Storage consists of an algorithm (pPosition) that determines the position of the first hinted block of the next consequent group to be prefetched from HDDs to the SSDs. At the beginning of the program execution, next consequent group to be prefetched from HDDs to the SSDs will be the first one. Initially, the application on-demand requests are found either in the HDD or in the SSD levels. This continues until the hinted data blocks arrive in the uppermost level(i.e., SSD). At this point, the application on-demand requests will be satisfied from the SSD level. As more I/O bandwidth becomes available for Eco-Storage to fetch hinted blocks, a more prefetching depth requests can be handled in the uppermost level. When the prefetched data blocks are completely read by the application's on-demand requests, they will be consumed from the SSD's prefetching buffer and new Eco-Storage prefetching requests are initiated. As we mentioned, more I/O bandwidth will increase the Eco-Storage prefetching depth which opens more room for the HDD to sleep and to save power.

The hybrid storage system consists of an array of HDDs and SSDs. $T_{hdd-cache}$ is the time spent in retrieving a single data block from the HDD to the application cache. $T_{ss-cache}$ is the time spent in retrieving a single data block from the SSD to the application cache. T_{hdd-ss} is the disk read latency from the HDD to SSD. $T_{ss-cache}$ is less than $T_{hdd-cache}$, because the performance of SSD is higher than that of HDD.

4.2 The pPosition (Starting Position) Algorithm

Eco-Storage prefetch the data from the HDD level to the SDD one in parallel with the application's on-demand I/O reading requests. At the beginning of the application execution, Eco-Storage issues a prefetching request for future hinted data blocks stating from a position that is calculated by pPosition algorithm. The depth of the prefetching request depends on the number of the prefetching groups (groups count is at least = 1). Each group's depth is equal to P_{depth} value that is calculated by Eq. 1. When all the prefetched data blocks are requested by the application, they will be consumed from the prefetching buffer in the SSD after they are fully fetched from the SSD to the cache (main memory) of the application. At this time, Eco-Storage issues a new prefetching request of the same number of groups for future hinted data block. The pPosition algorithm will determine the position of the next hinted data block to start prefetching from relatively from the last prefetched data block. So, the pPosition (starting position) is the elapsed distance between the beginning (or between the last recently prfetched data block) and the beginning or the next upcoming prefetching request (see: Fig. 2).

Algorithm 1 The pPosition Calculation Algorithm: Determines the next grouped prefetching request's starting position. The prefetching request will be from the HDD to the SSD

```
accesstime = 0
flag = true
blockcounter = 0
latestAppRequestPos = The position of the application's last on-demand requested data block
while flag = true do
  blockcounter ++
  accesstime + = T_ss-cache
  if accesstime ≥ T_hdd-ss then
    pPosition = latestAppRequestPos + blockcounter
    flag = false
    return pPosition
  end if
end while
```

Algorithm 1 Returns the hinted block position for which Eco-Storage begins/continues prefetching from. It depends on the T_{hdd-ss} assuming that all the on-demand data blocks are found in the SSD (fastest). It is calculated relatively from the position of the application's last on-demand requested data block. The algorithm calculates the hinted data block's position that will be accessed after enough time to have it read from HDD to SSD.

Figure 2 illustrates an example of Eco-Storage working with an application that contentiously issues on-demand I/O reading requests when using a single or double prefetching groups. Max_{BW} is set to 15. Eco-Storage prefetch P_{depth} number of data blocks per group from the HDD to the SSD.

The application finds the prefetched data blocks in the SSD. When the number of prefetching groups is increased, more consequent on-demand requests will be found in the SSD. This enables the HDD to standby and to save power. If the number of Eco-Storage prefetching groups equals to one, HDD will sleep for $T_{ss-cache} * pGroupsCount * P_{depth}$ time duration. On the other hand, if the number of Eco-Storage prefetching groups is more than one, HDD will sleep for $(T_{ss-cache} * pGroupsCount * P_{depth}) - ((pGroupsCount - 1) * T_{hdd-ss})$ time duration.

4.3 The Eco-Storage Algorithm

Algorithm 2 Eco-Storage Algorithm

```
pPosition = call the pPosition Calculation Algorithm
Input pGroupsCount value
Max_BW = Maximum number of concurrent reading requests
Pdepth = Max_BW − 1
while application on-demand requests do
  if all the recently prefeteched data blocks in the SSD
  are requested by the application then
    pPosition = call the pPosition Calculation Algorithm
    for block = pPosition To block = Pdepth * pGroupsCount do
      prefetch block to from HDD to SSD
      if bandwidth shortage then
        shrink the Pdepth value by 1
      end if
    end for
    if any prefetched data block is altered then
      discard the prefetched data block from the SSD
    end if
  end if
end while
```

Algorithm 2 The Eco-Storage algorithm calls the pPostion algorithm to determine the first hinted block to be fetched from HDD. It inputs the number of prefetching groups. Next Eco-Storage calculates the prefetching depth (i.e., Pdepth), which is affected by the available I/O bandwidth. Then, Eco-Storage keeps prefetching the data from the HDD to the SSD in groups in parallel with the application's on-demand requests. When the prefetetched group(s) is/are accessed by the application, Eco-Storage issue a new grouped prefetching request. Eco-Storage keeps calculating the pPostion value that determines the first hinted block to start the next grouped prefetching request from. Eco-Storage can handle any bandwidth shortage or data block modification.

5 Performance Evaluation

In this section, we are going to simulate our Eco-Storage solution in order to provide performance evaluation. First, we are going to validate the system parameters of our simulators using data collected from real-world storage systems Then, we are going to present our simulation results.

5.1 System Parameters' Validations

In this paper, we are going to use system parameters that are related to the storage system performance and that we have validated in [1]. Then, we are going to validate the parameters that are related to the energy consumption using the vendors specifications' documents of our research lab's test bed equipments.

The following list summarizes the validated system parameters:

- Data block size.
- T_{hdd-ss}: Time to fetch a single data block from HDD and to store the block in SSD.
- $T_{hdd-cache}$: Time to fetch a single data block from HDD and to store the block in the cache.
- $T_{ss-cache}$: Time to fetch a single data block from SSD and to store the block the cache.
- HDD_{Active}: HDD Power consumptions in active mode
- HDD_{Sleep}: HDD Power consumptions in standby mode
- SSD_{Active}: SSD Power consumptions in active mode
- SSD_{Idle}: SSD Power consumptions in idle mode

5.1.1 System Setup

All the system parameters used in our simulator and that are related to the storage system performance are validated by the testbed in our laboratory at Auburn. The following are storage devices tested in our laboratory:

- Memory: Samsung 3GB RAM Main Memory.
- HDD : Western Digital 500GB SATA 16 MB Cache WD5000AAKS.
- SSD: Intel 2Gb/s SATA SSD 80G sV 1A.

For energy related system parameters, we refereed to above testbed vendors documents.

In [1], we set the block size to be equal to 10MB because SSD is guaranteed to exhibit better performance than HDD at that data block size. We also validated SSD and HDD read latencies (i.e., $T_{ss-cache}$, $T_{hdd-cache}$ respectively). We also validate the T_{hdd-ss} latency which is the time spent in reading a data block from an HDD and write the block back to the SSD.

For energy related system parameters, we validated both HDD and SSD power consumptions in both active and

sleep/idle cases: HDD_{Active}, HDD_{sleep}, SSD_{Active}, and SSD_{Idle}. In the next few sections, we are going to discuss our system parameters' validations in details.

5.1.2 Block Size

As we mentioned, our preliminary results [1] based on our storage devices indicate that an SSD is guaranteed to exhibit better performance than HDD when the data block size is 10 MB for our single node system testbed. So, we are going to set the block size to 10 MB.

5.1.3 Storage System Performance Parameters' Validation

Storage system performance parameters that we are going to validate are the I/O access latencies like T_{hdd-ss}, $T_{hdd-cache}$, and $T_{ss-cache}$ when block size is 10MB.

According to [1], Figs. 3, 4, and 5 show that T_{hdd-ss} approximately equals to 0.122 seconds, $T_{hdd-cache}$ is about 0.12 seconds , and $T_{ss-cache}$ is in the neighbourhood of 0.052 seconds when block size is 10 MB. Worst case scenario is used by taking the highest recorded values.

5.1.4 Energy Consumption Parameters' Validation

According to our research lab testbed vendors specifications' documents [38, 39], our Western Digital HDD consumes 3.7 W/sec on read/writes (when it is in the active mode) and 0.5 W/sec when it is in the standby mode. Our Intel SSD consumes 150 mW/sec when active and 100 mW/sec when idle. This show that SSD are more energy efficient. Please note that we are going to assume no HDD spin up/down overheads since they are not mentioned in [38, 39]. Even if there are HDD spin up/down overheads, they will not make a significant increase in the system's power consumption due to the long standby intervals as we will see later.

5.2 The Eco-Storage Simulation

We implement the Eco-Storage algorithm in a trace-driven simulator written in C++. Power consumption in terms of (watt) when different cases of Max_{BW} and groups count is our performance metric that we used to evaluate Eco-Storage performance in the simulated two-level (i.e., SSD and HDD) storage system (see Fig. 1). For each testing case, we vary the numbers of Max_{BW} or the prefetching groups count. The figures below show the Eco-Stroage-enabled case. However, we found that without Eco-Storage, the power consumption is always constant. As we assumed previously, data blocks are initially placed in HDDs. Eco-Storage groups the informed prefetching process and sends copies of the original data from the lowest level (i.e HDD) to the uppermost one (i.e SSD). This aims to give the HDD the longest possible time to standby (sleep) and save power. The Eco-Storage mechanism coordinates the prefetching process of the hinted blocks from HDDs to SSDs. When the application issues an on-demand request for a prefetched data block, it will be found in the SSD.

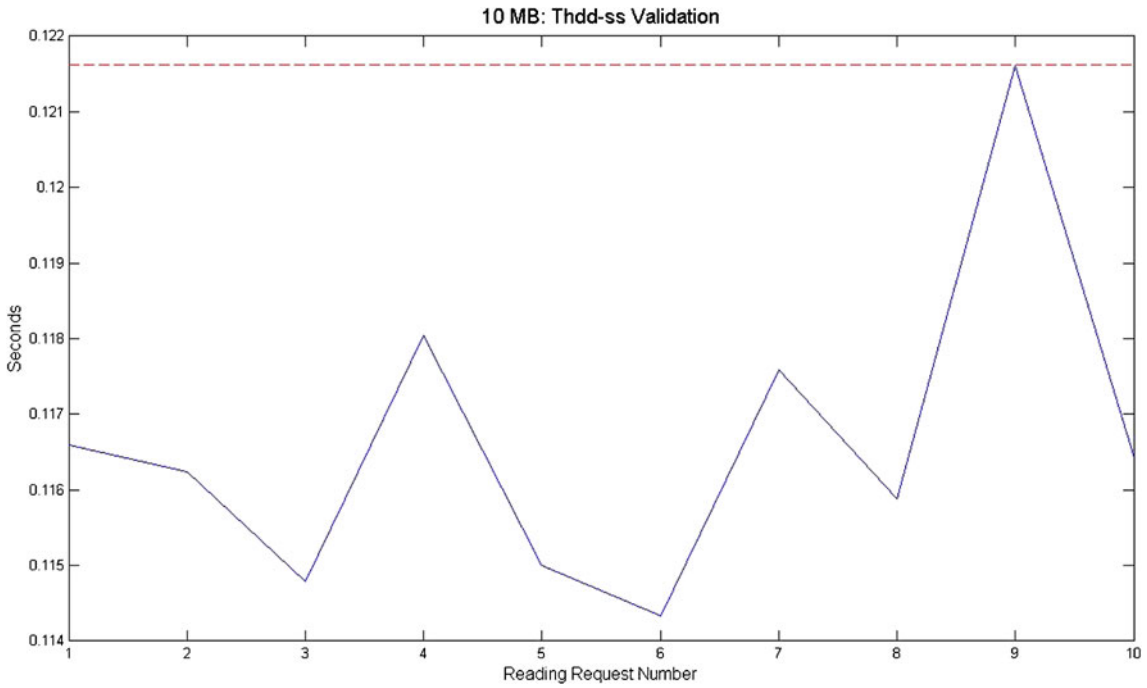

Figure 3 10 MB: Estimated T_{hdd-ss} is 0.122 s.

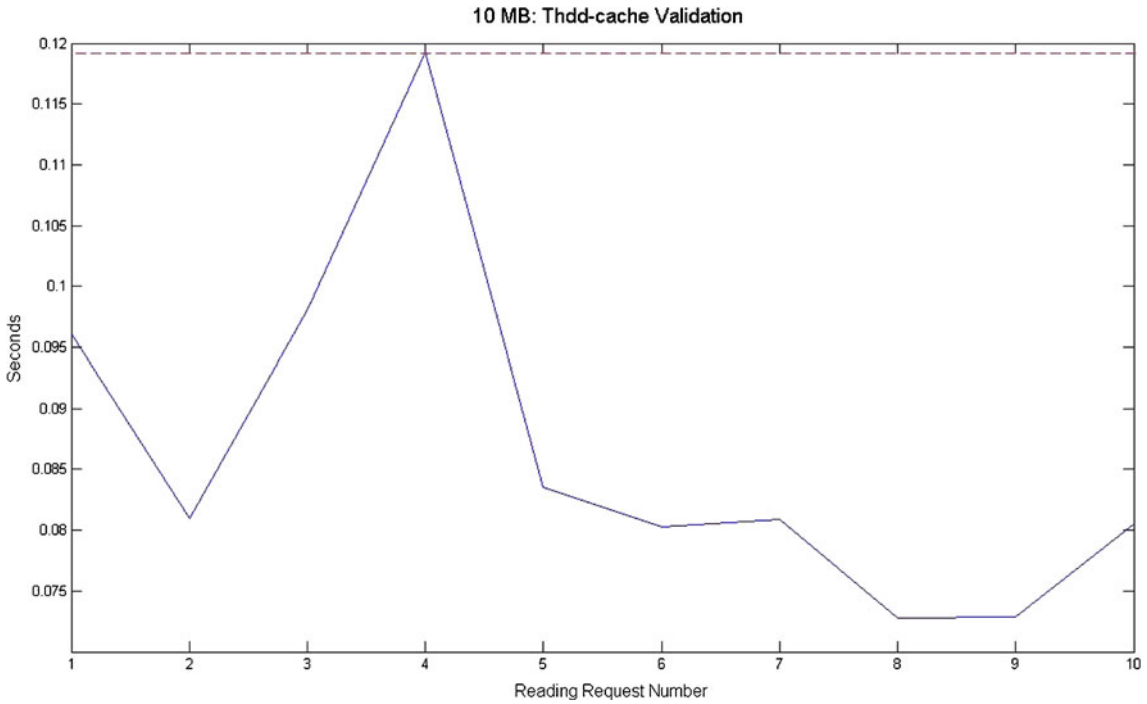

Figure 4 10 MB: Estimated $T_{hdd-cache}$ is 0.12 s.

In [1], we also used both LASR Machine01 and LASR Machine06 from the LASR trace suite [36, 37]. In this simulation, we are going to use LASR Machine06 trace because it has a longer execution time. LASR Machine06 consists of 51206 I/O read system calls. Each I/O request is accessing a data block from the two-level storage system. In our simulation, each of these I/O reading requests represents both an application I/O on-demand request and a future

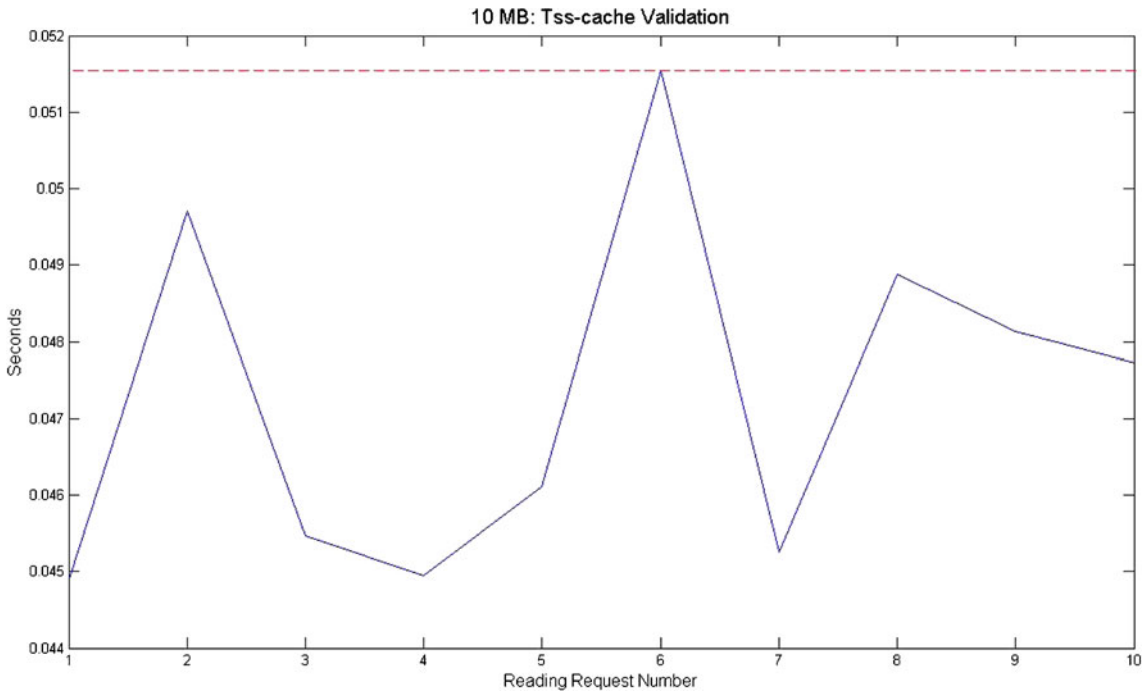

Figure 5 10 MB: Estimated $T_{ss-cache}$ is 0.052 s.

data block access hint for Eco-Storage informed prefetching mechanism.

In this simulation, we will use the system parameters that we validated in the previous subsection. Recall that the block size equals to 10MB. Data is initially stripped in the HDD layer of the two-level storage system.

Since both Eco-Storage and the application's on-demand requests work concurrently, the maximum number of read requests issued to HDDs is Max_{BW}. In this simulation, we simulate Eco-Storage starting from Max_{BW} value that equals to 15. Note that this value is 15 in the TIP study [3] since they are using 15 disks assuming that each single read request needs one disk not to face any congestions. Then, we simulate an increasing values of Max_{BW} one- by-one up to 20. The application on-demand request will use one slot of the maximum available bandwidth Max_{BW}. The rest are used for Eco-Storage prefetching.

The Eco-storage prefetching depth does not consume the SSD space. For example, when Max_{BW} equals to 15 and groups count equals to one, Eco-Storage prefetching depth is equal to 14. In this case, Eco-Storage needs 140 MB.

5.2.1 Reducing Energy Consumption

The experimental results show that Eco-Storage reduces the total energy consumption. In non-Eco-Storage case, the system's power consumption is always constant and equals to 23349.9 watt. Figures 6, 7, and 8 show the total energy consumption value in terms of (watt) with different numbers of prefetching groups count from 1 to 5 and when setting the Max_{BW} to subsequent values from 15 to 20. In all cases, Eco-Storage provides at least 75 % reduction in power consumption. However, when the prefetching groups count increases, Eco-Storage will become more efficient as it will become more able to increase the HDD standby (sleeping) time intervals. When Max_{BW} increases, Eco-Storage will become able to increase the prefetching group's depth which boosts it efficiency. Figures 9, 10, and 11 show the total energy consumption value in terms of (watt) with a variant numbers of Max_{BW} from 15 to 20 and when setting prefetching groups count to different values from 1 to 5. As we mentioned, increasing the Max_{BW} values will make Eco-Storage able to prefetch the data aggressively. When the groups count increases from 1 to 2, there will be a significant decrease in the total power consumption. This is because the amount of the prefetched data starts to duplicate at that point.

6 Conclusion & Future Work

In this paper, we presented an energy-aware informed prefetching technique - Eco-Storage- for hybrid storage systems. Eco-Storage makes use of the application-disclosed-

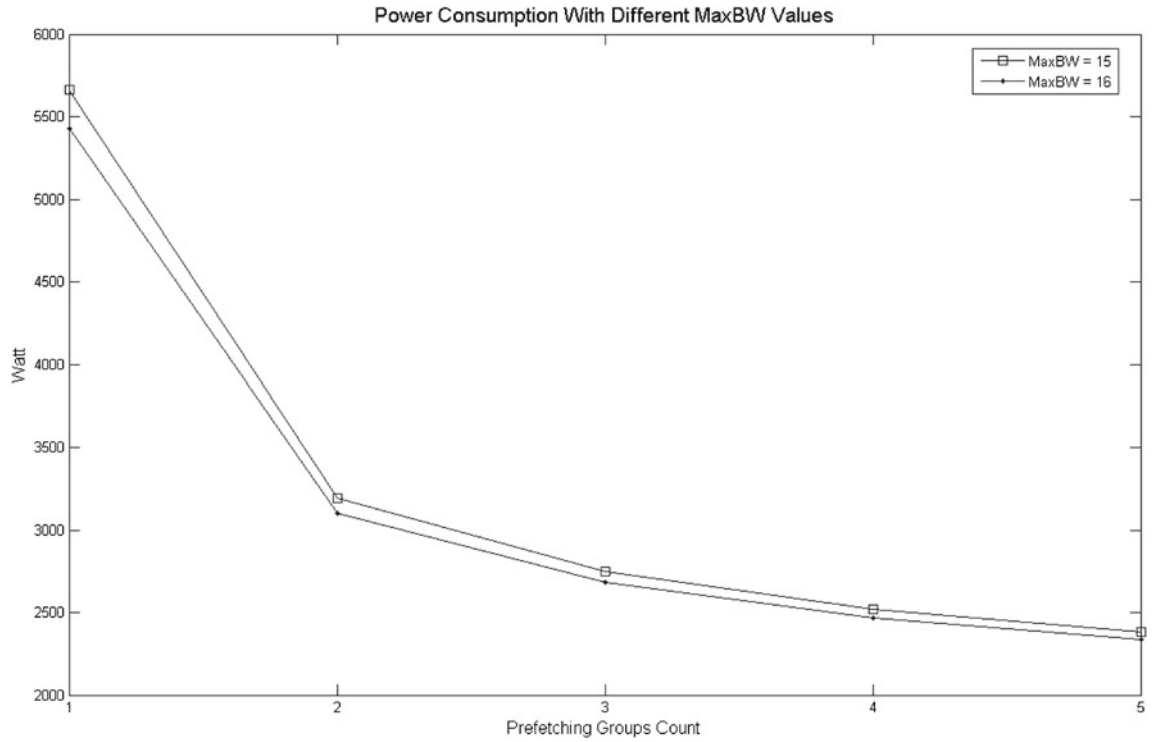

Figure 6 Total energy consumption in (watt) when using different number of prefetching groups count from 1 to 5. Max_{BW} is set to 15 and 16.

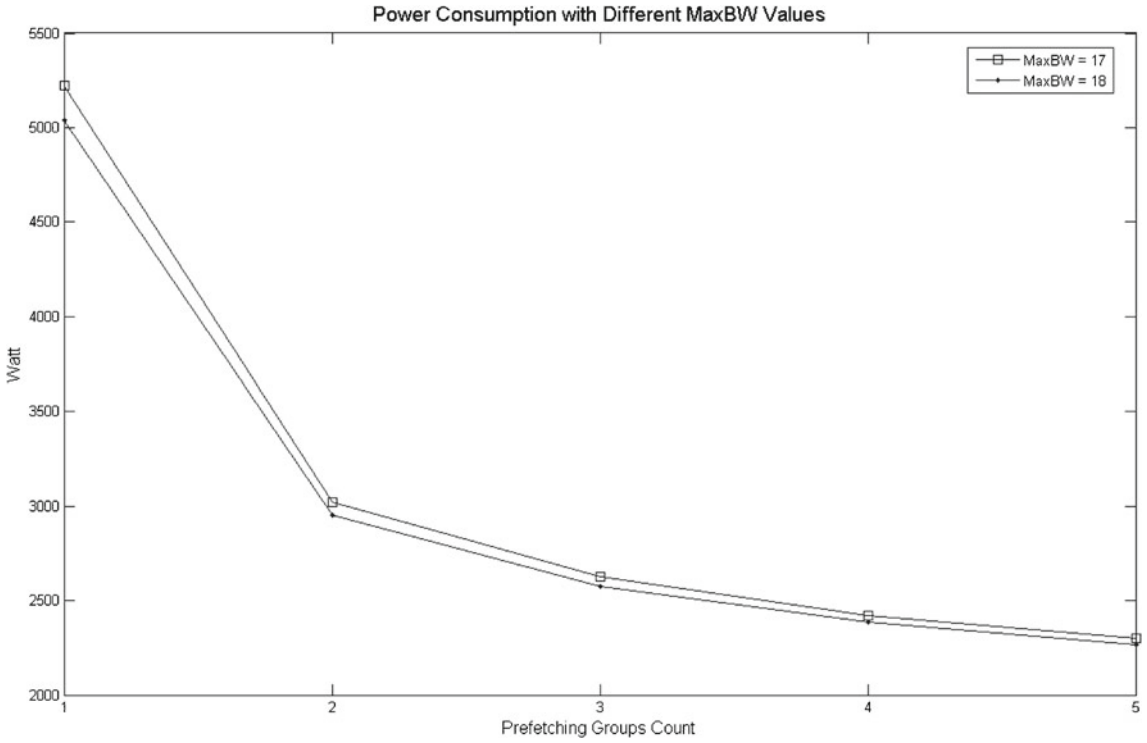

Figure 7 Total energy consumption in (watt) when using different number of prefetching groups count from 1 to 5. Max_{BW} is set to 17 and 18.

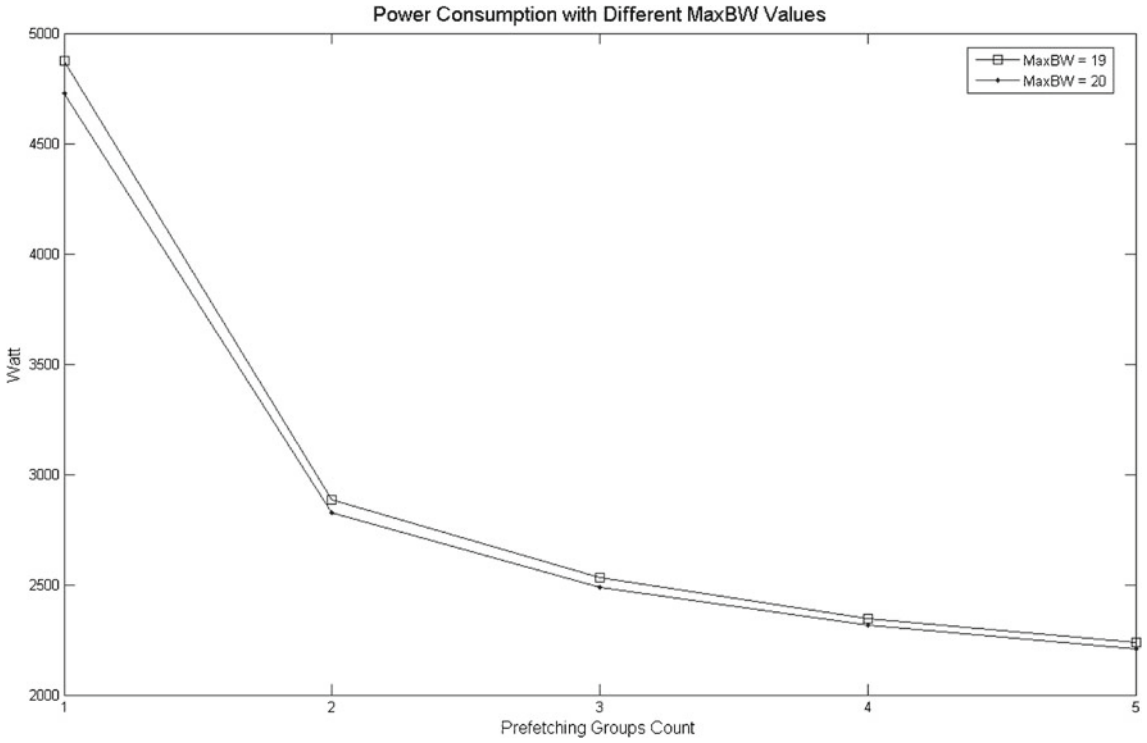

Figure 8 Total energy consumption in (watt) when using different number of prefetching groups count from 1 to 5. Max_{BW} is set to 19 and 20.

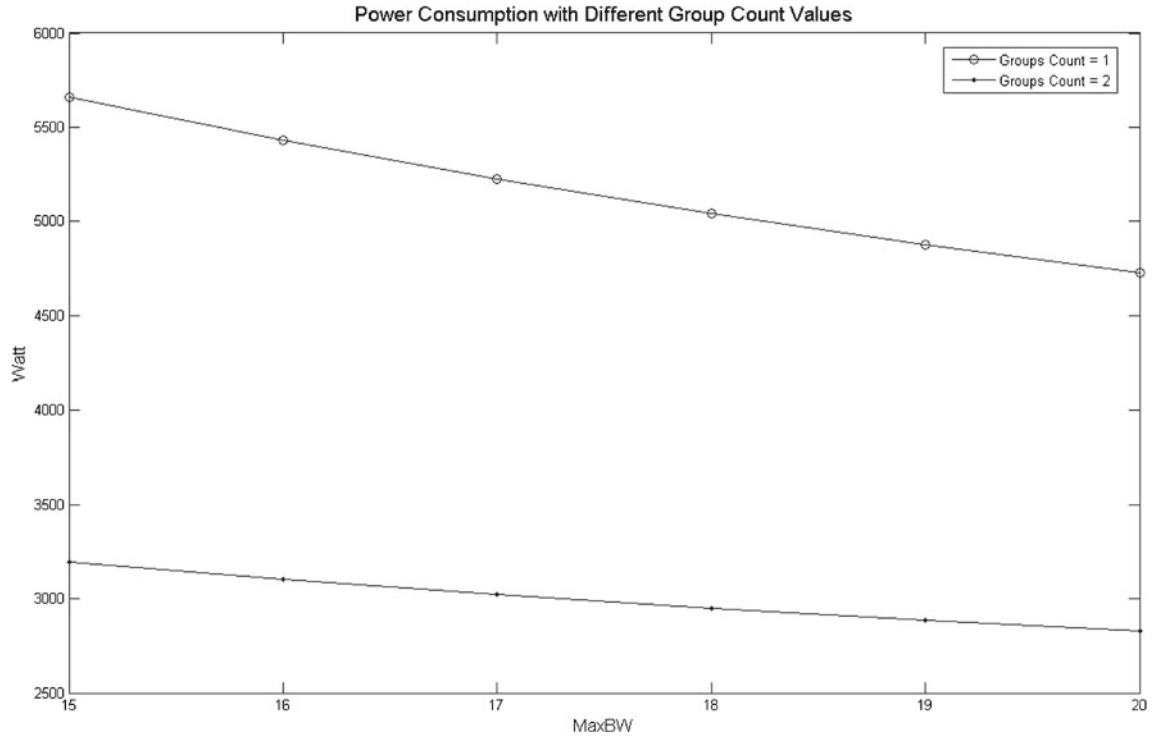

Figure 9 Total energy consumption in (watt) when using different number of Max_{BW} from 15 to 20. Prefetching groups count is set to 1 and 2.

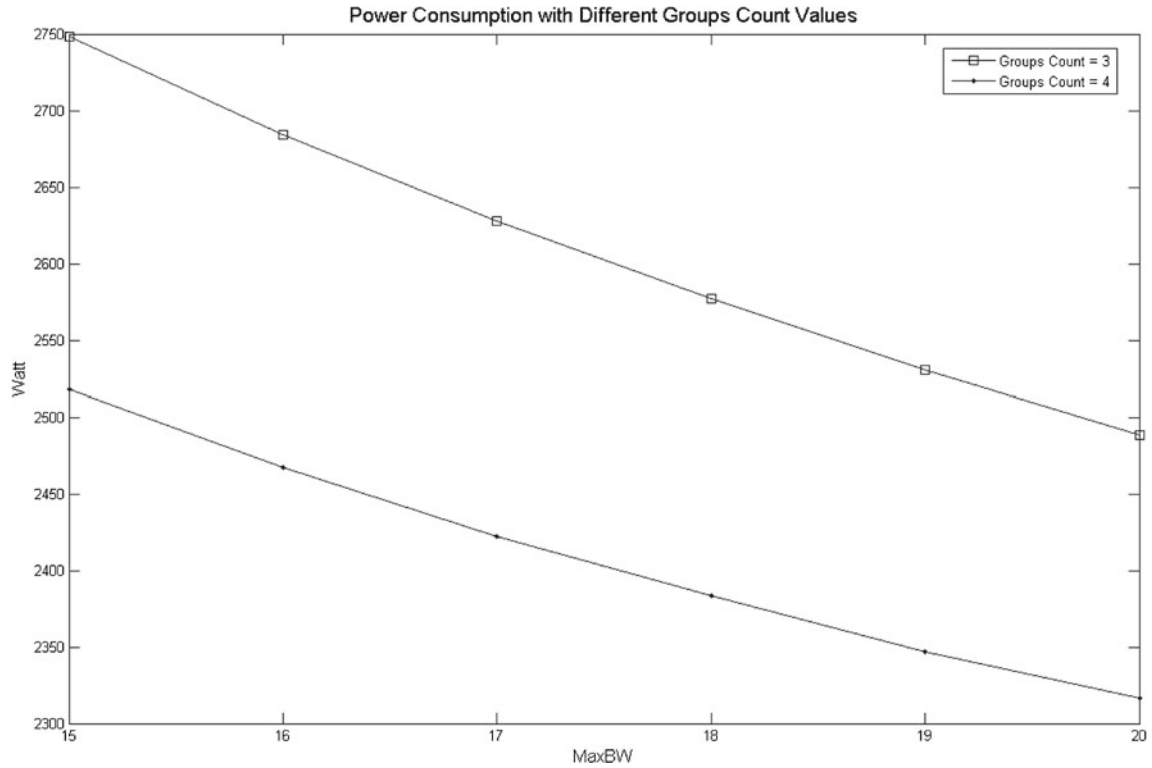

Figure 10 Total energy consumption in (watt) when using different number of Max_{BW} from 15 to 20. Prefetching groups count is set to 3 and 4.

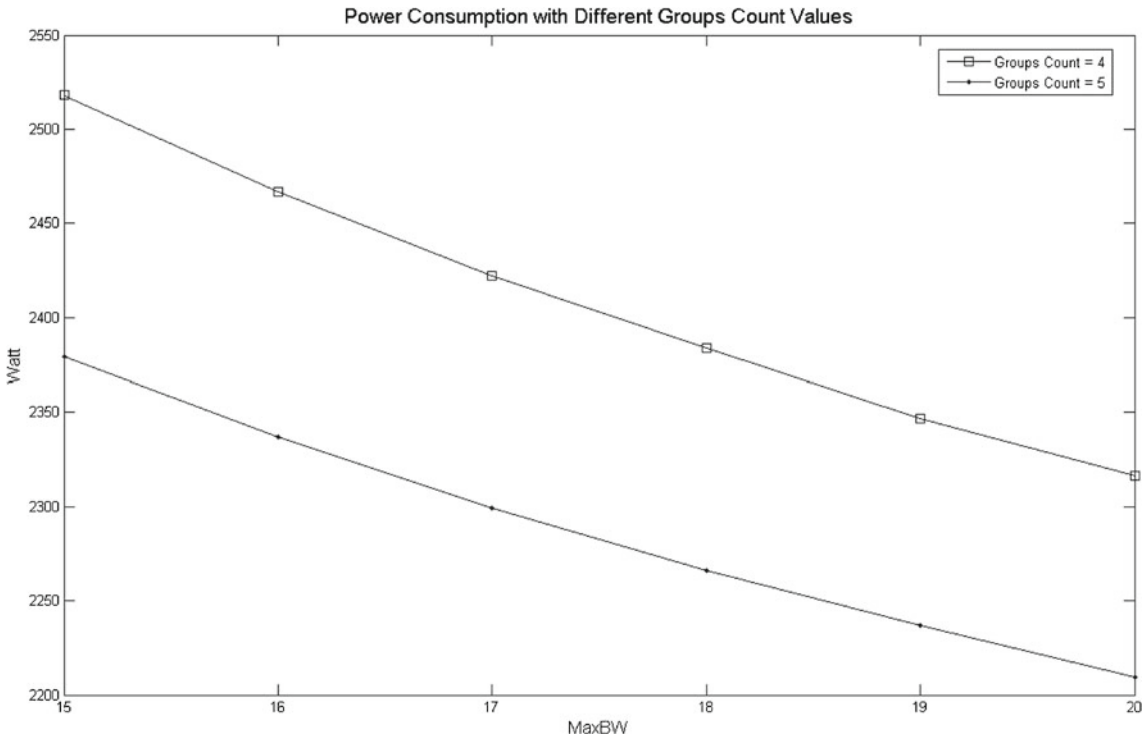

Figure 11 Total energy consumption in (watt) when using different number of Max_{BW} from 15 to 20. Prefetching groups count is set to 4 and 5.

hints to prefetch the hinted data to an energy efficient storage device (i.e. SSD) in the hierarchy in order to save power. The prefetching process is done in groups and in parallel with the application's on-demand I/O reading requests. Since SSD's are more energy efficient than HDDs, Eco-Storage aims to make the HDDs to standby as much as possible to reduce their power consumption. Compared with the existing energy-aware prefetching techniques, our Eco-Storage approach has the following two salient features. First, Eco-Storage works in hybrid storage systems. Next, it performs grouped informed prefetching in parallel with the application on-demand requests.

We implemented a simulator for our energy-aware informed prefetching schemes. Our results show that our Eco-Storage reduces the power consumption by at least 75 % under I/O-intensive workload conditions.

Our solution was tested when only using a single application. In reality, there maybe multiple applications running concurrently. In that case, our solution will stay valid because it will divide the aggregated bandwidth among the running applications and will stay able to prefetch the data in groups and to save energy. However, the expected performance improvement maybe degraded as the number of running applications increases. But in the same time, parallel storage systems can provide very high bandwidth thanks to their ability to scale up. Our future work is to test our solution when using several applications running concurrently.

Our future work also is to develop an energy-aware predictive prefetching approach for hybrid storage systems.

Acknowledgments This work was discussed with my colleagues in Dr. Xiao Qin's research group. I really thank them very much.

References

1. Al Assaf, M.M. Informed prefetching in distributed multi-level storage systems. http://hdl.handle.net/10415/2935.
2. Al Assaf, M.M., Al Ghamadi, M., Jiang, X., Zhang, J., Qin, X. (2012). A pipelining approach to informed prefetching in distributed multi-level storage systems. In *11th IEEE international symposium on network computing and applications (NCA)*. Cambridge.
3. Patterson, R.H., Gibson, G., Stodolsky, D., Zelenka, J. (1995). Informed prefetching and caching. In *Proceedings of the 15th ACM symposium on operating system principles* (pp. 79–95). CO.
4. Griffioen, J., & Appleton, R. (1994). Reducing file system latency using a predictive approach. In *Proceedings of the 1994 USENIX annual technical conference* (pp. 197–207). Berkeley.
5. Zhang, Z., Lee, K., Ma, X., Zhou, Y. (2008). PFC: Transparent optimization of existing prefetching strategies for multi-level storage systems. In *Proceedings of 28th international conference on distributed computing system* (pp. 740–751). Beijing.
6. Huizinga, M., & Desai, D.S. (2000). Implementation of informed prefetching and caching in linux. In *Proceedings of the international conference on information technology* (pp. 443–448). Las Vegas.
7. Patterson, R.H., Garth, A., Gibson, M. (1993). Satyanarayanan: a status report on research in transparent informed prefetching. *ACM SIGOPS Operating Systems Review*, *27*(2), 21–34.

8. Tomkins, A., Patterson, R.H., Gibson, G. (1997). Informed multi-process prefetching and caching. In *Proceedings of the 1997 ACM SIGMETRICS international conference on measurement and modeling of computer systems* (pp. 100–114). Seattle.
9. Chang, F., & Gibson, G.A. (1999). Automatic I/O hint generation through speculative execution. In *Proceedings of the third symposium on operating systems design and implementation* (pp. 1–14). New Orleans.
10. Patterson, R.H., Gibson, G.A., Satyanarayanan, M. (1992). Using transparent informed prefetching (TIP) to reduce file read latency. In *Proceedings of conference on mass storage systems and technologie* (pp. 329–342). Greenbelt.
11. Jason, F. (2002). Multi-level memory prefetching for media and stream processing. In *IEEE international conference on multimedia and expo* (pp. 101–104). St. Louis.
12. Kang, J., & Sung, W. (2001). A multi-level block priority based instruction caching scheme for multimedia processors. In *Proceedings of the 24th international conference on distributed computer systems* (pp. 125–132). Antwerp.
13. Nijim, M. (2010). Modelling speculative prefetching for hybrid storage systems. In *2010 IEEE fifth international conference on networking, architecture and storage (NAS)* (pp. 143–151). Macau.
14. Nijim, M., Zong, Z., Qin, X., Nijim, Y. (2010). Multi-layer prefetching for hybrid storage systems: algorithms, models, and evaluations. In *39th international conference on parallel processing workshops.* doi:10.1109/ICPPW.2010.18.
15. Kimbrel, T., Cao, P., Felten, E., Karlin, A., Li, K. (1996). Integrated parallel prefetching and caching. In *Proceedings of the 1996 ACM SIGMETRICS international conference on measurement and modeling of computer systems* (pp. 262–263). PA.
16. Patterson, R.H., & Gibson, G. (1994). Exposing I/O concurrency with informed prefetching. In *Proceedings of the third international conference on on parallel and distributed information systems* (pp. 7–16). Austin.
17. Chen, Y., Byna, S., Sun, X., Thakur, R., Gropp, W. (2008). Exploring parallel I/O concurrency with speculative prefetching. In *Proceedings of the 2008 37th international conference on parallel processing* (pp. 422–429). Portland.
18. Lewis, J., Alghamdi, M.I., Assaf, M.A., Ruan, X.-J., Ding, Z.-Y., Qin, X. (2010). An automatic prefetching and caching system. In *Proceedings of the 29th international performance computing and communications conference (IPCCC).*
19. Chen, Y., Byna, S., Sun, X., Thakur, R., Gropp, W. (2008). Hiding I/O latency with pre-execution prefetching for parallel applications. In *Proceedings of the 2008 ACM/IEEE conference on supercomputing* (pp. 1–10). Austin.
20. Byna, S., Chen, Y., Sun, X.-H., Thakur, R., Gropp, W. (2008). Parallel I/O prefetching using MPI file caching and I/O signatures. In *Proceedings of the ACM/IEEE conference on Supercomputing*. Austin.
21. Borthakur, D. (2008). HDFS architecture. The Apache Software Foundation. http://hadoop.apache.org/common/docs/r0.20.0/hdfs_design.pdf.
22. Shafer, J., Rixner S., Cox, A.L. (2010). The Hadoop distributed filesystem: balancing portability and performance. In *IEEE international symposium on performance analysis of systems & software (ISPASS)* (pp. 122–133). White Plains.
23. Hughes, G., & Murray, J. (2005). Reliability and security of RAID storage systems and D2D archives using SATA disk drives. *ACM Transactions on Storage*, *1*(1), 95–107.
24. Ganger, G.R., Worthington, B.L., Hou, R.Y., Patt, Y.N. (1994). Disk arrays: high-performance, high-reliability storage subsystems. *Computer*, *27*, 30–36. doi:10.1109/2.268882. ISSN:0018-9162. Ann Arbor.
25. Mishra, S.K., & Mohapatra, P. (1996). Performance study of RAID-5 disk arrays with data and parity cache. In *Proceedings of 1996 international conference on parallel processing* (pp. 222–229). Ithaca.
26. Baek, S.H., et al. (2001). Reliability and performance of hierarchical RAID with multiple controllers. In *Proceedings of twentieth annual ACM symposium on principles of distributed computing* (pp. 246–254).
27. Meador, W. (1989). Disk array systems. In *Proceedings of COMPCON spring 89. Thirty-fourth IEEE computer society international conference (pp. 143–146)*. San Francisco.
28. Hou, R., Menon, J., Patt, Y. (1993). Balancing I/O response time and disk rebuild time in a RAID5 disk array. In *Proceedings of on systems sciences* (pp. 70–79).
29. Drapeau, A., & Katz, R. (1993). Striped tape arrays. In *Proceedings of 12th IEEE symposium on mass storage systems*.
30. Yang, C.K., Mitra, T., Chiueh, T. (2002). A decoupled architecture for application-specific file prefetching. In *Freenix track of USENIX 2002 annual conference*.
31. Tuma, W. (2008). Comparison of drive technologies for high transaction databases. In *Storage developer conference*. Santa Clara. www.storage-developer.org.
32. Kaneko, T. (1974). Optimal task switching policy for a multilevel storage system. *IBM Journal of Research and Development*, *18*(4), 310–315.
33. Hadoop Archive Guide. http://hadoop.apache.org/mapreduce/docs/r0.21.0/hadoop_archives.html.
34. Papathanasiou, A.E., & Scott, M.L. (2004). Energy efficient prefetching and caching. In *Proceedings of the annual conference on USENIX annual technical conference* (pp. 22–22). Boston.
35. Manzanares, A., Qin, X., Ruan, X., Yin, S. (2011). PREBUD: prefetching for energy-efficient parallel I/O systems with buffer disks. *ACM Transactions on Storage (TOS)*, *7*(1), 1–29. doi:10.1145/1970343.1970346.
36. Lasr trace machine01. http://iotta.snia.org/traces/list/Subtrace?parent=LASR+Traces.
37. Lasr trace machine06. http://iotta.snia.org/traces/list/Subtrace?parent=LASR+Traces.
38. Western Digital HDD vendor specifications' document. http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-771438.pdf.
39. Intel SSD vendor specifications' document. http://www.intel.co.uk/content/www/uk/en/solid-state-drives/ssd-320-specification.html.
40. OCZ Technology Group, Inc SSD vs HDD comparison. http://www.ocztechnology.com/ssdzone/ssd-vs-hdd-comparison.html.

Maen M. Al Assaf received BS degree in computer science from Applied Science University, Amman, Jordan in 2006 and MA degree in Computer and Network Security from DePaul University (Jordan Campus) in 2008 and the Ph.D. degree in Computer Science from Auburn University, Al in 2011. Currently, he is an assistant professor of Computer Science in King Abdullah II School for Information Technology - The University of Jordan, Amman, Jordan. His research focus is distributed operating systems. He is a member of IEEE, ACM, and ASIST.

Xunfei Jiang is a PhD student in Computer Science and Software Engineering, Auburn University. She received the BS. and M.S. degrees in Computer Science from Huazhong University of Science and Technology (HUST), China, in 2004 and 2007. Then she joined Digital Video Networks Co., Ltd. in 2007 and Cisco Systems (Shanghai) Video Technology Co., Ltd in 2010. Her research interests include storage systems.

Mohamed Riduan Abid received a Ph.D in Computer Science in 2010 from Auburn University,USA. He received in 2006 the Excellence Fulbright Scholarship. He is currently an Assistant Professor of Computer Science at Alakhawayn University, Morocco.

Xiao Qin received the BS. and M.S. degrees in computer science from the HUST, China, and the Ph.D. degree in computer science from the University of Nebraska-Lincoln, Lincoln, in 1992, 1999, and 2004, respectively. Currently, he is an Associate Professor with the Department of Computer Science and Software Engineering, Auburn University. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation. He received the U.S. NSF Computing Processes and Artifacts Award and the NSF Computer System Research Award in 2007 and the NSF CAREER Award in 2009. He is a senior member of the IEEE.